Overview of Simulator for Project 1, CISC 340

By Abenezer Ayana and Austin Hippe

For this part of the project, we had to create a C program that takes in machine code and produces the desired output, like adding registers 1 and 2 and storing that value in register 3. For our simulator, we create a struct named state_struct that holds the state of our machine. In it we have 3 ints, pc, num_memory, and instructionCount, for our program counter, how much memory we have (how many lines are in the machine code), and how many instructions we have completed. We also have int pointers mem and reg that are int arrays that represent our registers and our memory. We also have the print_state function that prints each part of the machine, and the print stats function which prints the total number of executed instructions. The first thing we do is get the machine code using GNU getopt. Once we do that, we read in the machine file to see how many lines it has. This is the amount of memory we have, and it is used to create space for our mem array. After we have done this, we use malloc to create space for both of our arrays in state_struct, and we set all values to 0. Once we do this, we print the state of the machine, and then we fill our mem array with the machine code from the file. Once we do this, we then go into a for loop which does the reading and produces the desired output. The first line in our loop adds 1 to our instructionCount variable, which is how many instructions have been done so far. After this, we call simulateAsm, which is the function that does the instructions for us. It takes 2 parameters, and these parameters are a pointer to the struct itself, and an int that holds the machine code to be processed. In the function, the first thing we do is get the opcode out of the machine code. We do this by shifting the variable machineCode to the right 22 spots. Once we have this opcode, then we can identify what instruction we want to complete. After this we have 8 if statements, each one for each instruction we have. For 0 and 1 (add and nand), we right shift to pull out register A, register B, and the destination register. After we do this, we set the destReg variable equal to the value in register A plus or nand the value in register B. After this, we go to the actual register in our state_struct and put the value we calculated into destReg. For lw we shift to get our destIndex, regA, and we use the convert_num function to convert the 16-bit offset to a 32-bit number. Once we do this, we calculate what value in memory we want to put in the destination register and put it in destReg, and then use destIndex to to put that value into the specified register. For sw we shift to get regA, regB, and offset, and go to regB + offset in memory and store the value in regA. Again, we use the convert_num function to convert offset to a 32-bit number. For beq, we do the same shifting to get regA, regB, and offset, using the convert_num function on the offset. This time we compare and see if the value in regA is equal to the value in regB, and if it is, we add 1 to our instructionCount, set our pc to pc + 1 + offset, and call simulateAsm again. For jalr we shift to get regA and regB, add 1 to our pc, store the pc in regA, and set our pc to the value of regB. For halt, we use print_state and print_stats to print out the final state and stats of the machine and exit the program. For noop, we obviously do nothing. After we call the simulateAsm function in the for loop, we call print_state to print the state after the ran instruction and add 1 to our pc. Some difficulties we had was getting the data out of the registers for some of the calculations like add and nand instead of just adding the number of the registers, and we also had some difficulty implementing the instructionCount. After going through our test cases, it looks like everything is up and working correctly.